CARIBOO : An Induction Based Proof Tool for Termination with Strategies –Extended version–

Olivier Fissore, Isabelle Gnaedig, Hélène Kirchner
LORIA-INRIA & LORIA-CNRS
B.P. 239 F-54506 Vandœuvre-lès-Nancy Cedex
Phone : + 33 3 83 58 17 00
Fax : + 33 3 83 27 83 19
e-mail: fissore@loria.fr,gnaedig@loria.fr,Helene.Kirchner@loria.fr

ABSTRACT

We describe Cariboo, the implementation of an inductive process recently proposed to prove termination of rewriting under strategies on ground term algebras. The method is based on an abstraction mechanism, introducing variables that represent ground terms in normal form, and on narrowing, schematizing reductions on ground terms. It applies in particular to non-terminating systems which are terminating with innermost or local strategies. The narrowing process, well known to easily diverge, is controlled by using appropriate abstraction constraints. The abstraction mechanism lies on satisfiability of ordering constraints. Thanks to the power of induction, these ordering constraints are often simple and automatically solved by our system. Otherwise, they can be treated by the user or any external automatic solver. On many examples, Cariboo even enables to succeed without considering any constraint at all ; the process is then completely automatic. Cariboo offers a graphical view of the proof process. It is implemented in ELAN, a rule based programming environment, and so can be used for proving termination of ELAN programs.

Keywords : rewriting, termination, innermost, local strategy, rule based language, induction, narrowing, ordering constraint, ELAN.

Link to the system :
http://www.loria.fr/~fissore/Demo/description.html.

1. INTRODUCTION

Termination of rewriting is a crucial problem in automated deduction, for equational logic, as well as in programming, for rule based languages. The property is important in itself, but it is also required to decide properties like confluence and sufficient completeness, or to allow proofs by consistency.

In this paper, we present a tool addressing the termination problem in the context of proof environments for rule-based programming languages, such as ASF+SDF [15], Maude [5], Cafe-OBJ [10], or ELAN [4], where a program is a term rewriting system and the evaluation of a query consists in rewriting a ground expression.

In such a context, it is useful to have termination proof tools working in a finer way than most of the existing tools, which prove in general termination of standard rewriting (rewriting without strategy) on the free term algebra. Our proof tool, named Cariboo (for **C**omputing **A**bst**R**action for **I**nduction **B**ased termination pr**OO**fs), allows to prove termination under specific reduction strategies, which is interesting when the computations diverge for standard rewriting, and to prove termination on the ground term algebra, which is interesting in particular for term rewriting systems that are not terminating on the free term algebra.

As far as we know, specific termination proof techniques for rewriting with strategies have only been given for the innermost case [1] and for the context sensitive rewriting [17, 18, 11] on free term algebras. Here, we propose an implementation of a general inductive mechanism for proving termination of rewriting under strategies, based on an explicit induction on the termination property, and design it for the innermost and local strategies.
Our proof tool is implemented in ELAN for ELAN : integrated into the ELAN environment itself, it makes it possible for the user to prove the termination of ELAN programs. It can obviously be used for any other rule based program.

We first introduce the basic mechanisms of our inductive proof principle for the innermost rewriting, and then present the corresponding implementation in ELAN. Then we explain how our proof principle has been designed for rewriting with local strategies. Using Cariboo is described through an example, for the innermost case.

2. PROOF PRINCIPLES

2.1 Induction for termination

To prove innermost termination of a TRS on the ground term algebra $\mathcal{T}(\mathcal{F})$, the main intuition is to observe the innermost rewriting derivation tree starting from any ground term t. Proving innermost termination on ground terms amounts to prove that all innermost rewriting derivation trees have only finite branches.

For proving that a ground term t innermost terminates, we proceed by induction on the ground term algebra with an ordering $\succ$ stable by context and having the subterm property, assuming that for any t' such that $t \succ t'$, t' innermost terminates. More precisely, we first prove that a basic set of minimal elements for $\succ$ innermost terminates. As the sub-

term property for $\succ$ is required, the set of minimal elements is a subset of the set of constants of the TRS. We then simulate the innermost rewriting derivation trees starting from any instance of $g(x_1, \ldots, x_m)$, for all defined symbols g of the signature $\mathcal{F}$, for proving that all their branches are finite.

The derivation trees of the ground terms of the form $g(t_1, \ldots, t_m)$ are simulated by a proof tree starting from $g(x_1, \ldots, x_m)$, a term of $\mathcal{T}(\mathcal{F}, \mathcal{X})$, the free term algebra, where $x_1, \ldots, x_m$ are variables, and developed in alternatively using two main concepts, namely narrowing and abstraction. Narrowing schematizes all rewriting possibilities of the terms, according to the value of their variables. The abstraction process simulates the innermost normalization of subterms in the derivations. It consists in replacing these subterms by special variables, taken in a new set $\mathcal{N}$ and called NF-variables, representing any normal form of these subterms. This abstraction step is performed on subterms that can be assumed innermost terminating by induction hypothesis.

2.2 Building proof trees

So the proof process amounts to develop abstract trees whose nodes are composed of a current term of the free term algebra $\mathcal{T}(\mathcal{F}, \mathcal{X} \cup \mathcal{N})$, and a set of ground substitutions appropriately represented by constraints. Each node schematizes a possibly empty set of ground terms, namely all instances of the current term that are ground solutions of the constraints.

Let us study the derivation tree of any ground term $t = g(t_1, \ldots, t_m)$. As said before, the innermost rewriting relation is simulated by two mechanisms :

- first, the t_i are supposed to be innermost terminating by induction hypothesis, and replaced by *abstraction variables* X_i representing respectively any of their normal forms $t_i\downarrow$: these variables, the NF-variables, will only be instantiated by terms in normal form. Reasoning by induction allows us to only suppose the existence of the $t_i\downarrow$ *without explicitly computing them.* The abstraction of the term t_i by the variable X_i is memorized by the equality $t_i\downarrow = X_i$, called *abstraction constraint.* This step is the abstraction step ;
- second, narrowing the resulting term $g(X_1, \ldots, X_m)$ at position ϵ in all possible ways into terms u : its ground instances $g(t_1\downarrow, \ldots, t_m\downarrow)$ are indeed either irreducible, or only reducible at the top position ; this is the narrowing step. Note that the narrowing process, well known to easily diverge, is controlled by using the information expressed by the abstraction constraints. Indeed, a narrowing step is valid only if its narrowing most general unifier (mgu) satisfies them.

Then the termination problem of t is reduced to the termination problem of the terms u. If $t \succ u$, by induction hypothesis, u is supposed to be innermost terminating. Otherwise, one can apply a similar reasoning on $u = f(u_1, \ldots, u_n)$. We then have to assume that $(t \succ t_1, \ldots, t_m) \wedge (t \succ u_1, \ldots, u_n)$. This process is iterated until getting either a term t' such that $t \succ t'$, or such that t' is irreducible. In these cases, any innermost derivation starting with $t \rightarrow_R^+ t'$ is finite.

The proof tool, introduced in Section 4, integrates this mechanism and builds the proof trees automatically.

2.3 The induction ordering

The induction ordering is not given a priori, but is more and more constrained along the proof by imposing ordering constraints between terms that must be comparable. Observe that this process can fail on $g(u_1, \ldots, u_n)$ if we cannot decide that the ordering constraint $(t > t_1, \ldots, t_m) \wedge (t > u_1, \ldots, u_n)$ is satisfiable. Then, we cannot conclude anything.

An important point is that the induction ordering $\succ$ has to be the same for all proof trees. Observe also that the noetherian property of $\succ$ is implied by the stability by context and the subterm property.

As we develop proof trees from terms with variables $g(x_1, \ldots, x_m)$ instead of reasoning with every ground term, the proof process involves ordering constraint solving of the form $g(x_1, \ldots, x_m) > v$. An ordering constraint $(t > t')$ is satisfiable if there exists an ordering $\succ$ and at least one instantiation (ground substitution) θ such that $\theta t \succ \theta t'$. We say that $\succ$ and θ satisfy $(t > t')$.

Constraint solving is discussed in Section 2.6.

2.4 Modularity of the method

The strength of our method is that it is general enough to be efficiently combined with other termination proof mechanisms. Indeed, the termination hypothesis is made on subterms to be infered on terms. If this hypothesis cannot be ensured by the induction process itself, it is sometimes possible to establish the innermost termination of the subterms by another way. In the following we use a predicate $TERMIN(u)$ that is true iff every ground instance of u innermost terminates.

As an example, for establishing that $TERMIN(u)$ is true, in some cases, the notion of usable rules, introduced in [1], can be used. Given a TRS $\mathcal{R}$ on $\mathcal{T}(\mathcal{F}, \mathcal{X})$ and a term $u \in \mathcal{T}(\mathcal{F}, \mathcal{X} \cup \mathcal{N})$, we determine a calculable superset of the rewrite rules that are likely to apply to any of its ground instances, for the standard rewriting relation, until its ground normal form is reached, if it exists. Proving termination of any ground instance of u then comes down to proving termination of its usable rules, which is often much easier than orienting the whole TRS. The usable rules can be proved terminating with classical termination methods like simplification orderings, or with our inductive method itself. An interesting point here is that the ordering that can be used to prove termination of the usable rules, either with classical methods or inductively, is completely independent of the main induction ordering. Note also that for proving termination of the constants, usable rules can also be used. The notion of usable rules and their properties are fully developed in [13]. Our proof tool computes them automatically, when needed.

This flexibility of our method, allowing to integrate other termination proof approaches in the main inductive proof process in a natural way, gives us a formal framework for different termination tools to efficiently cooperate, unlike other methods that, when failing, require the user to find another proof technique by himself.

2.5 Inference rules and data structures

We now give a sketch of our proof procedure, fully explained in [13] and described by the following inference rules, working with sets C and A, collecting respectively the ordering constraints and the abstraction constraints :

1. **Abstract** for abstracting the immediate subterms of the current term. Note that there is no need to abstract the subterms that are ground terms in normal form or NF-variables. When a subterm t_i is abstracted into an NF-variable X_i, then the abstraction constraint $t_i\downarrow = X_i$ is added to the abstraction constraints set A, initially empty. A subterm t_i is abstracted :

   - either if the induction hypothesis applies on t_i ; an ordering constraint is added to the ordering constraints set C, initially empty ;
   - or if $TERMIN(t_i)$ can be shown.

   Note also that if t_i is not narrowable and if its variables are NF-variables, then any of its ground instance is in normal form, and then t_i does not need to be abstracted. This includes in particular the cases where t_i is either a ground term in normal form or an NF-variable ;

2. **StopA** applying when **Abstract** cannot apply on a subterm of the current term ;

3. **Narrow** for applying a narrowing step to the current term at the top position. The narrowing mgu σ used to narrow the current term is applied to A, which becomes σA. The narrowing mgu is valid only if σA is satisfiable (see Section 2.6). This rule is applied with all possible rewrite rules ;

4. **StopN** applying when the current term is not narrowable any more ;

5. **Stop** applying when the current term t can be assumed to be innermost terminating, either by induction hypothesis or if $TERMIN(t)$. If the induction hypothesis applies on the current term t, then an ordering constraint is added to C.

These inference rules work on states consisting of the current term, the abstraction constraint set A and the ordering constraint set C. These states represent the nodes of the proof tree, as introduced in Section 2.2. The rules **Stop** and **StopN** replace the current state $(\{t\}, A, C)$ by $(\emptyset, A, C)$, standing for a successful state, while the rule **StopA** replaces $(\{t\}, A, C)$ by $(\sharp, A, C)$, standing for a failure state. The proof tree is said to be successful if every branch ends with a successful state.

The inference rules are repeatedly applied so that the application of **Abstract** is followed by all possible applications of **Narrow**. After each narrowing step, we attempt to end the current branch of the derivation tree with a success state by applying **Stop**. When **Abstract** cannot apply, then **StopA** applies, which ends the current branch of the proof tree with a failure state. When **Narrow** does not apply, **StopN** applies, which ends the current branch of the proof tree with a success state. The strategy expressed in ELAN is given in section 3.2.

2.6 Constraints solving

The abstraction constraints set A is said to be satisfiable if there exists at least one ground substitution θ such that for each conjunct $u\downarrow = v$ of A, we have $\theta u\downarrow = \theta v$.

The constraint problem (A, C) is satisfied by an ordering $\succ$ if A is satisfiable, and for all instantiations θ satisfying A, $\succ$ and θ satisfy C (i.e. all constraints of C). (A, C) is satisfiable if A is satisfiable and there exists an ordering $\succ$ as above.
Hence, deciding the satisfiability of (A, C) would require to express all instantiations satisfying A. An interesting point of our method is that we do not need to characterize all those instantiations. It is enough to exhibit one of them to prove the satisfiability of A. In such a case, if $\succ$ is required to be stable by substitution (the induction ordering is then a simplification ordering), a sufficient condition for the ordering $\succ$ to satisfy (A, C) is that $t \succ t'$ for each constraint $t > t'$ of C.

For each node of the derivation tree, the set of possible instances of the current term is given by the ground substitutions characterized by A and C : they are the ground substitutions θ satisfying A such that an ordering $\succ$ and θ satisfy C.

In any case, the ordering constraints are automatically generated by Cariboo. On many examples, they are satisfied by the subterm ordering. Otherwise, then can be delegated to automatic existing ordering constraints solvers like [6, 12].

2.7 Example

We give here an example to show what a derivation tree looks like. Moreover, the example is complete enough to emphasize some strong characteristics of the method.

Let $\mathcal{R}$ be the following TRS, defined on $\mathcal{F} = \{f, g, 0, 1\}$:

$$\begin{array}{ll} f(0,1,x) & \to f(g(x,x),x,x) \\ g(x,y) & \to x \\ g(x,y) & \to y \end{array}$$

Note that this example is not terminating ; indeed, we can get the following infinite rewrite chain :

$$f(g(0,1),g(0,1),g(0,1)) \to^1 f(0,g(0,1),g(0,1)) \to^2$$

$$f(0,1,g(0,1)) \to^\epsilon f(g(0,1),g(0,1),g(0,1)) \to^1 \ldots$$

However, this chain is not an innermost rewriting chain, and in fact, $\mathcal{R}$ is innermost terminating. We prove it by using our inductive proof process.

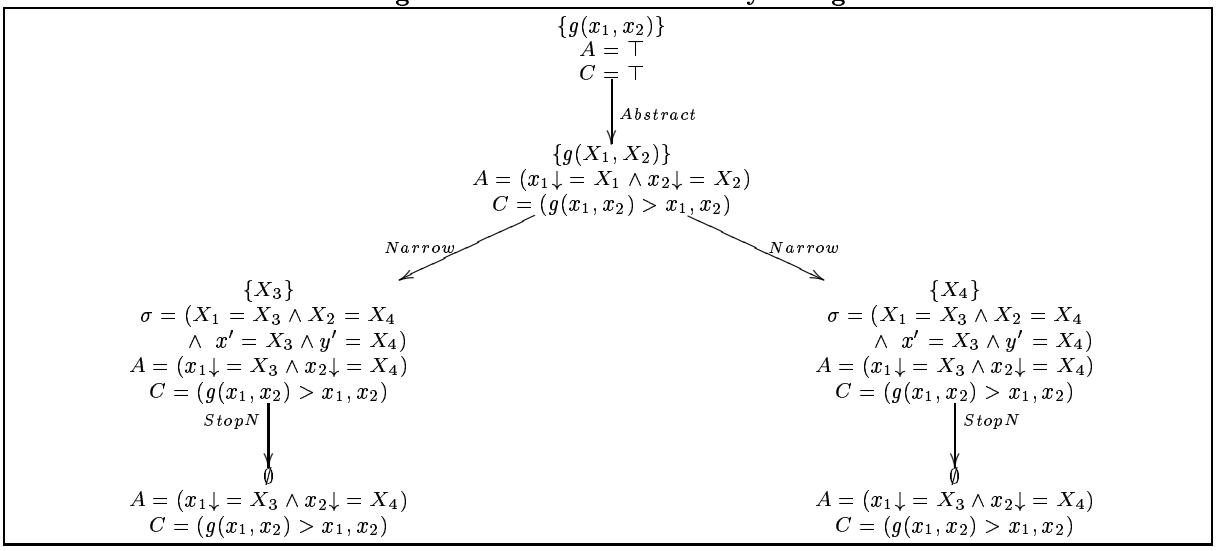
Obviously, the constants 0 and 1 are innermost terminating, since they are constructor symbols.

Here, we have two defined symbols : g and f. The proof tree whose root is $g(x_1, x_2)$ is given in Figure 1. The first inference rule we are trying to apply is **Abstract**, abstracting x_1 and x_2 into X_1 and X_2 either if the induction hypothesis applies, or if an other criterion of termination can be used. Since the induction ordering is supposed to have the subterm property and is stable by substitution, the induction hypothesis applies on x_1 and x_2 ; indeed, with such an ordering, any ground substitution θ satisfies the constraints $(g(x_1, x_2) > x_1, x_2)$, which are added to the set of ordering constraints. The current term $g(x_1, x_2)$ is then transformed into $g(X_1, X_2)$.

Then the narrowing is performed (for comments on the strategy, see the section 3.2) with the last two rewrite rules. We get, up to a renaming, X_3 and X_4, both abstraction variables, no more narrowable. Note that A is satisfiable by any instantiation θ such that $\theta x_1 = \theta x_2 = \theta X_3 = \theta X_4 = 0$.

The proof tree whose root is $f(x_1, x_2, x_3)$ is given in Figure 2. As previously, the first application of **Abstract** is made possible thanks to the subterm property of the induction ordering, stable by substitution. Then **Narrow** applies

Figure 1: Derivation tree for symbol g



with the first rule, thus giving only one branch. The constraint A is then satisfiable by any instantiation θ such that $\theta x_1 = \theta x_3 = \theta X_4 = 0$ and $\theta x_2 = 1$.

The last application of **Abstract**, abstracting the subterm $g(X_4, X_4)$, illustrates the modularity of the method : since X_4 is a variable whose any instance is a ground term in normal form, the usable rules of $g(X_4, X_4)$ (see Section 2.4), are the last two ones of the TRS $\mathcal{R}$. Since these two rules are orientable by any simplification ordering, any instance of the term $g(X_4, X_4)$ is bound to be innermost terminating, and we can abstract it by a new special variable X_5.

Another interesting point is that the term $t = f(X_5, X_4, X_4)$ is detected to be non narrowable any more, thanks to the satisfiability test of A. Indeed, one could think that the substitution $\sigma = (X_5 = 0 \wedge X_4 = 1 \wedge x'' = 0)$ would enable to rewrite the term σt at the top position, using the first rewrite rule of $\mathcal{R}$. However, X_5 is supposed to be the normal form of $g(X_4, X_4)$, and the mgu σ would contradict this fact. In other words, the abstracting constraint σA would not have been satisfiable. This illustrates the fact that each node of the derivation tree characterizes only a subset of the whole ground term algebra.

Figure 2: Derivation tree for symbol f

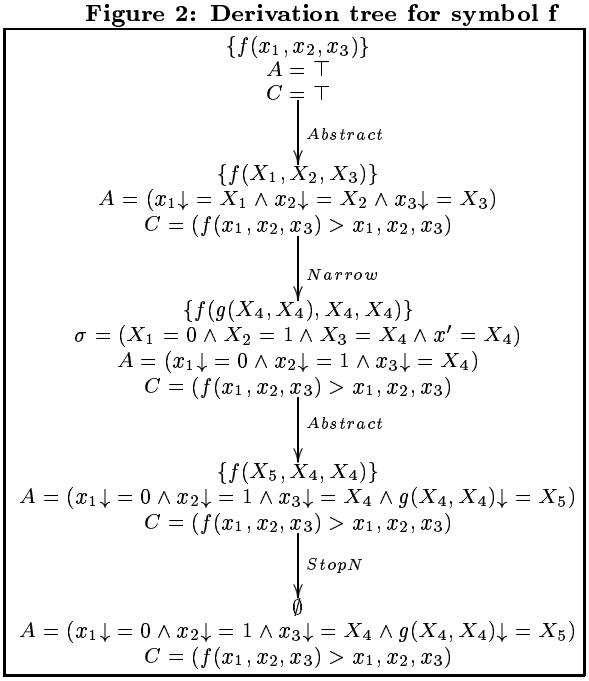
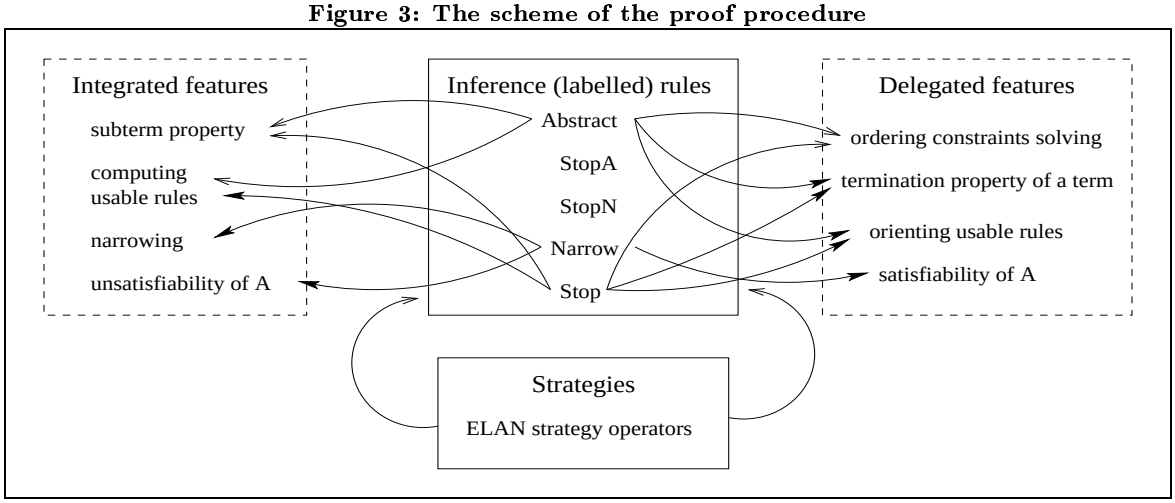


Figure 3: The scheme of the proof procedure



3. IMPLEMENTATION IN ELAN

In this section, we explain how our proof process is implemented. Let us first point out some characteristics of the ELAN language.

3.1 The ELAN language

The ELAN system [3] is an environment for specifying and prototyping deduction systems in a language based on rewrite rules controlled by strategies. It offers a natural and simple logical framework for the combination of the computation and deduction paradigms as it is backed up by the concepts of rewriting calculus and rewriting logic. It supports the design of theorem provers, logic programming languages, constraint solvers and decision procedures, and offers a modular framework for studying their combination.

An ELAN program consists of two kinds of rules : unlabelled rules, evaluated according to a built-in leftmost-innermost strategy, and labelled rules, whose evaluation is defined by the user, thanks to the strategy operators manipulating them.

Both a compiler and a parser are available for the ELAN language. The implementation of Cariboo requires quite a few input/output features. It has been written for the ELAN parser, and runs on the version 3.5.4 of ELAN.

An interesting point of our approach is its reflexive aspect : we study termination for rule-based languages and, in such a context, our proof tool written in ELAN can be applied to ELAN programs as well. The inference rules are implemented in a natural way in the ELAN rule-based formalism.

3.2 The inference rules and their application

The implementation of our proof process consists of three parts :

1. the data structures : terms, abstraction and ordering constraint sets ;
2. the inference rules : ELAN labelled rules manipulating the data structures ;
3. the strategy controlling the application of inference rules.

The rules **Abstract**, **StopA**, **Narrow**, **StopN** and **Stop** have been implemented in ELAN as labelled rules manipulating a data structure containing :

- the reference term (initial term of a derivation), used for the application of the induction hypothesis when needed ;
- the current term, together with the abstraction constraints, representing a set of ground terms ;
- the ordering constraints, incrementally set along the proof process ;
- some extra information, to build the tree and manage the renaming of the variables : the depth of the tree, the numbers of variables together with the number of NF-variables.

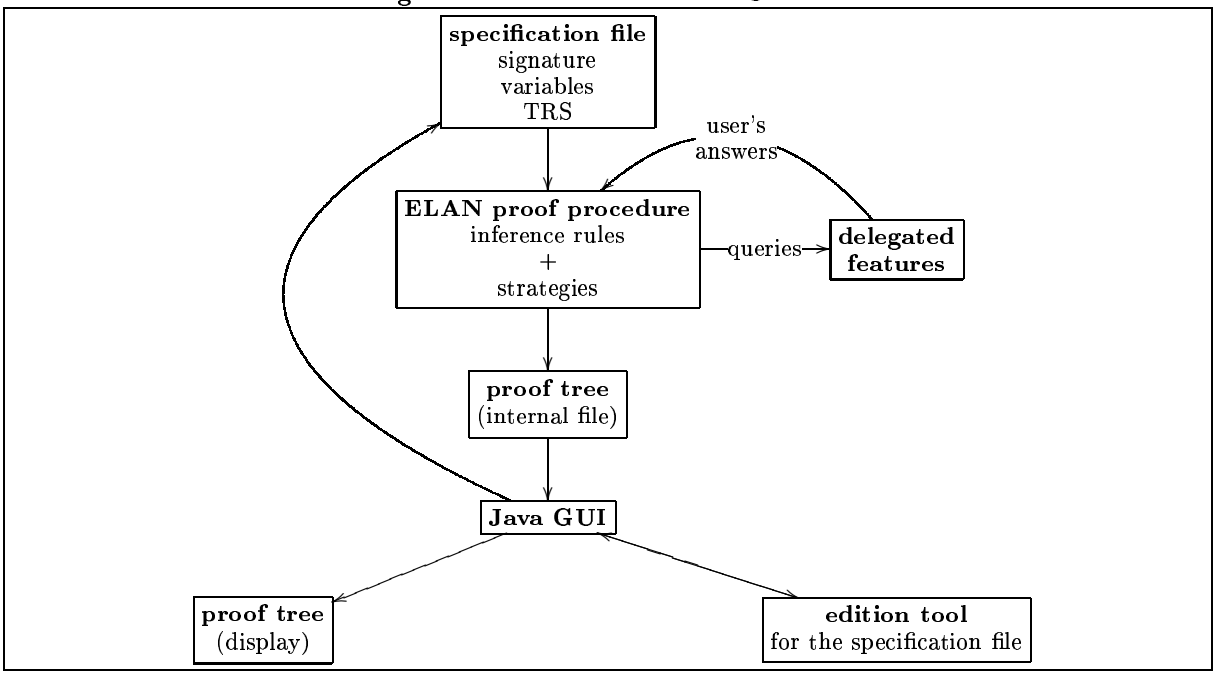
Since the inference rules are expressed in ELAN by labelled rules, they can be combined very easily by the programmer. The basic strategy manipulating these rules is the following. We first try **Abstract**, and apply **StopA** if it does not succeed ; this is performed by the ELAN strategy *first(Abstract, StopA)*. Then we apply **Narrow**, consisting in finding a possible narrowing step at the top position of the current term. This rule is applied in all possible ways, thanks to the *dk* operator. If there is no narrowing, then **StopN** applies : the corresponding ELAN strategy is then *first(dk(Narrow), StopN)*. Finally we attempt a **Stop** application, which tests whether the current term can be assumed to be innermost terminating, and iterate the process. If we denote *id* the identity strategy that has no effect but always succeeds, the ELAN strategy for the whole process is the following :

repeat(first(Abstract, StopA) ; first(dk(Narrow), StopN) ; first(Stop, id)).*

The procedure terminates when a success or failure state is reached for each branch ; indeed, on such states, the rules do not apply anymore. The success case is when every final state is a success state, the failure case when one of the final states is a failure one.

Moreover, the process can diverge if we have an infinite application of **Abstract** and **Narrow**. To make our proof

Figure 4: The architecture of Cariboo



tool terminate in this case, we have fixed a maximum value for the depth of the tree ; once this value is reached, then the ELAN rule *Abstract* fails, and *StopA* applies, ending the branch likely to be infinite with a failure state.

3.3 Integrated and delegated features

In this section we come back to the description of each inference rule (see section 2.5) from an operational point of view. Applying inference rules involves processes that can either be automatically computed by internal procedures, or else, that are let to the user or to an external tool. These processes are respectively called integrated features and delegated features. The scheme of the proof procedure with respect to these features is summarized on Figure 3.

- *Abstract* : the proof tool automatically checks, for each subterm t_i, whether it is narrowable. If it is not, then t_i has to be abstracted : the proof tool then computes the ordering constraints to be solved for applying the induction hypothesis. If they can be solved with the subterm ordering, then t_i is automatically abstracted. In particular, the first application of *Abstract* is automatic, since it always consists in solving an ordering constraint of the form $g(x_1, \ldots, x_m) > x_1, \ldots, x_m$. Otherwise, the proof tool computes the usable rules of t_i, whose termination has to be proved for $TERMIN(t_i)$. If these rules are orientable with the subterm property, then t_i is automatically abstracted. If they are not, then the proof tool delegates to the user the ordering constraints and the usable rules whose termination has to be proved, together with the term t_i to prove termination. As said before, the ordering constraints can then be delegated to any independent solver.

- *StopA* and *StopN* : these are just automatic rules that apply respectively if *Abstract* or *Narrow* fails.

- *Narrow* : all narrowing mgus are automatically computed by the unification algorithm provided by the ELAN library. Note that we only consider narrowing at the top position, thanks to the abstraction step performed just before.
  The proof process needs to know whether the narrowing mgu σ is compatible with the abstraction constraints set, that is whether σA is satisfiable. Our proof tool automatically detects the unsatisfiability of σA when the narrowing step assigns a reducible term to an NF-variable, instantiable only by ground terms in normal form. In the other cases, this test is delegated to the user.

- *Stop* : it just works like the *Abstract* rule, but with the whole current term instead of its subterms.

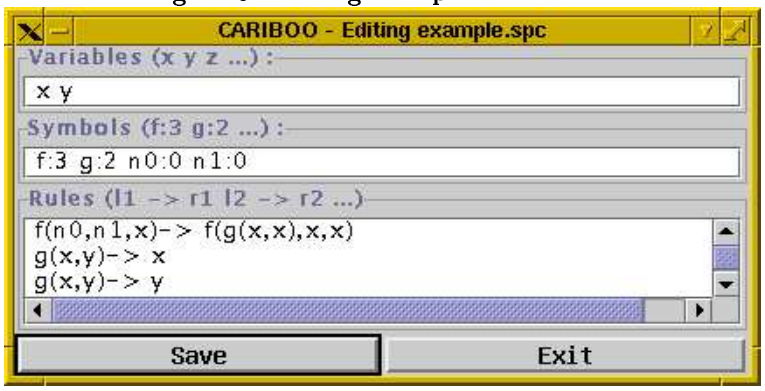
3.4 Automation versus success

An interesting point of our proof process is that its correctness is maintained even when all or some of the delegated features are suppressed. So, by suppressing some requirements, one may gain automation ; but, by doing so, one lacks information that may be necessary to make the process succeed. In this section, we propose different possibilities of suppressing delegated features and combine them to conciliate the automation of the proof process and its chances to terminate with success.

As a matter of fact, the less numerous the conditions to be checked, the more automatic the process. We may then first try to apply the rules without any of their delegated features (see Section 3.3 and Figure 3). By suppressing them, the proof tree may become larger :

- some branches may be longer, because we do not check any more whether the current term is terminating in the *Stop* rule ;

Figure 5: Editing the specification file



- since the satisfiability of A is no longer checked, we may also have some extra branches with an unsatisfiable abstraction constraints set A. In this case, A characterizes an empty set of ground instances, and the branches are merely useless, but computed (automatically, that is the point) anyway.

We then obtain a supertree of the tree we would obtain by using all delegated features ; by construction, if all branches of this supertree end with a successful state, then it will be the same for the smaller tree. Therefore the correctness of the proof process is preserved.
On the other hand, if a failure state occurs in the supertree, it may belong to a useless branch, and hence this does not mean that this failure state would occur in the smaller tree. Hence, in case of failure, one tries again the process with some delegated features.

So we can first try again the process with the delegated features of the *Abstract* rule, namely the ordering constraint satisfiability condition and the termination proof of a term. Since we may still obtain a supertree of the tree we would obtain without suppressing any feature, in case of failure, one can finally try again the process with, in addition, the delegated features of the *Stop* rule.

Note that since *Stop* is performed right after *Narrow*, one can move the satisfiability test of A from *Narrow* to *Stop*. In this case, the *Narrow* rule will be performed without any delegation.

The strategy implemented in Cariboo tries all attempts described above, from the most to the less automatic.

Finally, an alternative to this strategy, available for the user, is to postpone any constraint solving, by first assuming the constraints satisfiable : he then gets a termination result modulo their satisfiability.

4. THE PROOF TOOL

In this section, we present the external behavior of Cariboo. We first introduce the architecture of our proof tool, and then illustrate how to use it through the development of a complete example.

4.1 The architecture of Cariboo

Basically, as illustrated on Figure 4, the proof tool consists of two parts :

1. the ELAN proof procedure, generating the proof trees and writing information (constraints and proof trees) both on the standard output and in files ;
2. the graphical user interface (GUI), making it possible for the user to follow each step of the proof process : which defined symbols have already been treated and, for each of them, the proof tree together with the detail of each state.

The ELAN proof procedure expects the user to give the TRS which he wants to prove innermost termination of. More precisely, it expects, as shown in figure 5, the rewrite rules of the system along with the variables and the signature of the algebra.
An edition tool has been written in Java to help the user enter these data, which are then transformed into the ELAN specification used by the main program (see Figure 4).

During the process, the ELAN proof procedure builds the proof tree, and writes the detail of each state into a file. These information are then picked up by the Java GUI that displays in a graphical way the evolution of the proof process (see Figure 6).

To handle the delegated features, the proof procedure directly communicates with the user through the window in which it has been launched (see Section *4.2.4* and Figure 7).

Then, once the proof process is achieved, the proof tree of each term $f(x_1, \ldots, x_n)$ has been generated by the ELAN proof procedure for each defined symbol f and displayed by the Java GUI. Then we can get the detail of each state of these trees. If none of the proof trees contains a failure state (coming from a **StopA** application), then the TRS given as input is innermost terminating. The trace can be saved in text format.

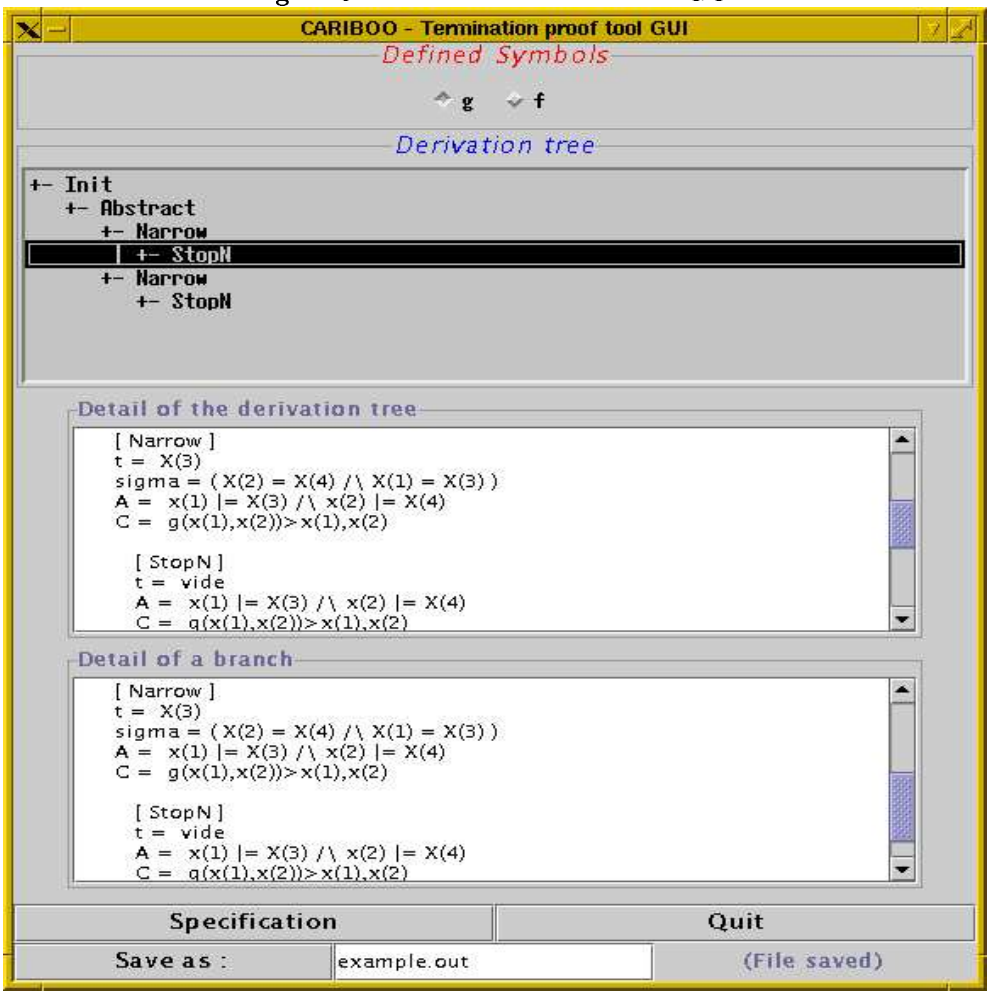
4.2 How to use Cariboo

In this section, we explain how to use the proof tool to establish innermost termination of a TRS. To achieve this, the example introduced in Section 2.7 is dealt from scratch.

4.2.1 Editing the TRS

As said before, we first have to create the specification file needed by the ELAN proof procedure. Let us remind that this file must contain the rules of the TRS together with the signature and the variables. One may build this specification file by hand, accordingly to the following format (the keywords *specification, Vars, Ops, Rules* and *end of specification* cannot be changed) :

Figure 6: The main window of the GUI




```
specification example

Vars
        x y
Ops
        f:3 g:2 n0:0 n1:0

Rules
        f(n0,n1,x)      -> f(g(x,x),x,x)
        g(x,y)          -> x
        g(x,y)          -> y

end of specification
```

Let us remark that in ELAN, the operators must begin with a letter ; this is why constructors 0 and 1 are denoted here $n0$ and $n1$.

If one does not want to edit the specification by hand, Cariboo offers the possibility to directly fill in the different fields of the specification file (Figure 5), provided the GUI is running (see Section *4.2.2*). To do so, the user has to click on the *Specification* button, in the main window of the GUI (figure 6). Then, saving the specification generates the same specification file as above.

4.2.2 Launching the ELAN proof procedure

Once the specification file is created, it can be used for the ELAN proof procedure :

```
elan cariboo example.spc
```

Once the parsing is done, the proof procedure expects a query from the user. If he wants the proof tool to deal with all defined symbols of the signature, he can simply enter the query *all end*. Otherwise, he can precise which symbols he wants to consider by giving a list of these symbols ; the syntax for a list of N elements in ELAN is the following : $elem1.elem2.\cdots.elemN.nil$.

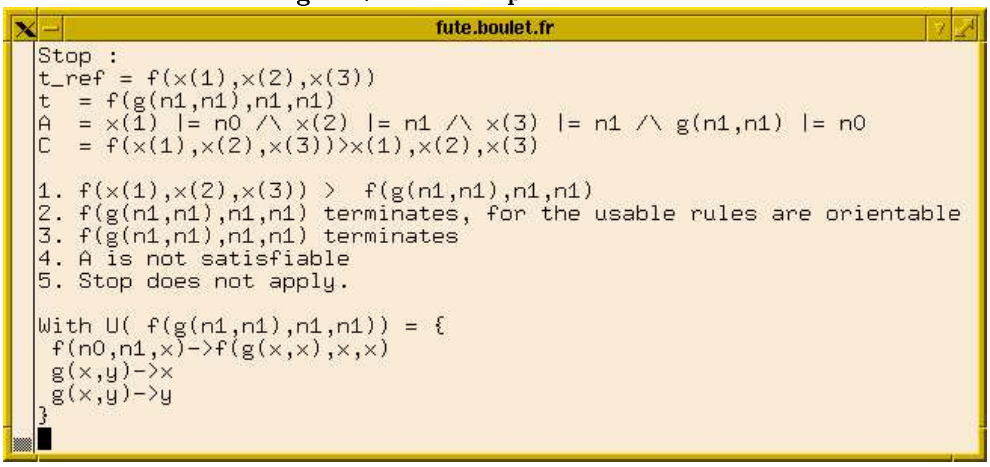
```
enter query term finished by the key word 'end':
all end
```

Then the ELAN proof procedure runs, until either termination or an interaction. For each specified defined symbol, the derivation tree is built.

4.2.3 Launching the GUI

Since the communication between processes is achieved through files, the Java GUI has to be launched from the same directory as the ELAN proof procedure, in which text files are created and shared. Once the user is in this working

Figure 7: An example of interaction



directory, he can launch the Java GUI by giving as parameter the name of the specification file containing the TRS (see Section *4.2.1*), whether this file already exists or not.

```
java CaribooGUI example.spc
```

Note that running the GUI is purely optional ; doing so gives information on the processing of the ELAN proof procedure, but has no effect on the procedure itself.

If the user chooses to launch the GUI, he then gets on the screen the window of figure 6 whose fields, initially empty, correspond to :

1. the **defined symbols** that appear are the ones treated (or being treated) by the ELAN proof procedure. Clicking on one of the defined symbols enables the user to have a look at :
2. the corresponding **derivation tree**, as defined in Section 2.2 and illustrated on the example of Section 2.7 ; for each derivation tree, we get :
3. the **detail of each state**, made visible with the current term, the abstraction constraint set A, the ordering constraint set C and the narrowing mgu in case of a *Narrow* application. In this window we get all states of the tree, while on the window below, we get :
4. the **detail of a particular branch**, by just clicking on the final state of the branch in the derivation tree window ;

Note that the display of the derivation tree is updated while the ELAN proof procedure is processing each symbol. Therefore each step of the process is made visible, at any moment.

4.2.4 *Interactions with the user*

When Cariboo requires an interaction with the user, the ELAN proof procedure interrupts for displaying a query and expecting an answer from the user.

The current example raises no interaction for the treatment of the symbol g, and two interactions for the treatment of the symbol f : a first unsucessful attempt to apply *Stop*, and a second attempt, sucessfull because the satisfiability test of A (moved from the previous application of *Narrow*) is negative (Figure 7).

4.2.5 *Saving the trace*

The figure 6 shows the GUI once the example has been completely treated. Its trace, that can be saved in textual format by using the *Save as* button, is the following :

```
Derivation tree for symbol g :

+- Init
  +- Abstract
     +- Narrow
     | +- StopN
     +- Narrow
       +- StopN

Detail :

[ Init ]
t_ref =  g(x(1),x(2))
A = true
C = true

   [ Abstract ]
   t =  g(X(1),X(2))
   A =  x(1) |= X(1) /\ x(2) |= X(2)
   C =  g(x(1),x(2))>x(1),x(2)

      [ Narrow ]
      t =  X(3)
      sigma = ( X(2) = X(4) /\ X(1) = X(3) )
      A =  x(1) |= X(3) /\ x(2) |= X(4)
      C =  g(x(1),x(2))>x(1),x(2)

         [ StopN ]
         t =  vide
         A =  x(1) |= X(3) /\ x(2) |= X(4)
         C =  g(x(1),x(2))>x(1),x(2)

      [ Narrow ]
      t =  X(4)
      sigma = ( X(2) = X(4) /\ X(1) = X(3) )
      A =  x(1) |= X(3) /\ x(2) |= X(4)
      C =  g(x(1),x(2))>x(1),x(2)

         [ StopN ]
         t =  vide
         A =  x(1) |= X(3) /\ x(2) |= X(4)
         C =  g(x(1),x(2))>x(1),x(2)

------------------------------------------------
```

```
Derivation tree for symbol f :

+- Init
   +- Abstract
      +- Narrow
         +- Abstract
            +- Narrow
               +- Cut

Detail :

[...]
```

Even if the GUI is not running, the trace can be saved through the command line :

```
java Sauvegarde example.spc
```

Note that in general, when the proof process converges, the derivation trees are not very deep.

5. LOCAL STRATEGIES ON OPERATORS

The strength of our inductive proof process is that it is generic enough to cover different rewriting strategies. Beyond the innermost strategy, we thus designed inference rules for the outermost strategy [13] and for local strategies on operators [9]. In the following, we describe the latter strategies.

We first remind the definition of the so-called *LS (rewriting) strategy*, as expressed in [14] and studied in [7], and then emphasize the differences between the ELAN proof procedure for the innermost and for the LS-strategy.

5.1 Definition of the LS-strategy

An *LS rewriting strategy* (or LS-strategy) on terms of $\mathcal{T}(\mathcal{F}, \mathcal{X})$ (resp. of $\mathcal{T}(\mathcal{F})$) is a function LS from $\mathcal{F}$ to the set of lists of integers $\mathcal{L}(\mathbb{N})$, defining a rewriting strategy as follows.

Given a LS-strategy such that $LS(f) = [p_1, \ldots, p_k]$, $p_i \in [0..arity(f)]$ for all $i \in [1..k]$, for some symbol f, normalizing a term $t = f(t_1, \ldots, t_m)$ with respect to $LS(f)$, consists in normalizing all subterms of t at positions $p_1, \ldots, p_k$ successively, according to the strategy. If there exists $i \in [1..k]$ such that $p_1, \ldots, p_{i-1} \neq 0$ and $p_i = 0$ (0 is the top position), then

- if the current term t' obtained after normalizing $t|_{p_1}$, $\ldots$, $t|_{p_{i-1}}$ is reducible at the top position into a term $g(u_1, \ldots, u_n)$, then $g(u_1, \ldots, u_n)$ is normalized with respect to $LS(g)$ and the rest of the strategy $[p_{i+1}, \ldots, p_k]$ is ignored,
- if t' is not reducible at the top position, then t' is normalized with respect to $p_{i+1}, \ldots, p_k$.

5.2 Comparison with the innermost strategy

First, let us remark that the leftmost innermost strategy can be expressed by means of LS-strategies. Indeed, it comes down to setting $LS(f) = [1, \ldots, n, 0]$ for each symbol f of arity n. Likewise, any other innermost strategy can be simulated by an appropriate LS-strategy.

In addition, termination of rewriting has been proved [16] to be equivalent for the leftmost innermost and any other innermost strategy (rightmost, ...). Therefore our inductive proof process for the LS-strategy holds for any innermost strategy.

Let us now explain in what the inference rules for the local strategies differ from the ones designed for the innermost strategy.

First, the process does not involve the abstraction of all subterms of a term $f(u_1, \ldots, u_m)$, but only of subterms pointed by the LS-strategy, and in a specified order. Then, given a strategy such that $LS(f) = [i_1, \ldots, i_p, 0, j_1, \ldots, j_q]$, we first try to abstract the subterms $u_{i_1}, \ldots, u_{i_p}$, before trying to narrow the resulting term at the top position. This gives rise to two fundamental differences with respect to the innermost proof process :

- during the proof process, the manipulated terms may contain both classical variables, instantiable by any term, and NF-variables, instantiable only by terms in normal form. This has to be taken into account in the narrowing process ;
- the rule *Narrow* narrows at the top position the term resulting from the abstraction step in all possible ways, with narrowing mgus σ. If these mgus do not cover all possible instances of the term, then the remaining instances must be reduced according to the positions occuring in the strategy after the position 0, by definition of the LS-strategy. Characterizing these instances goes through negation of substitutions, and then slightly different constraints involving disequations.

Finally, the definition of the LS-strategy generates some non conventional normal forms. Indeed, for a symbol f of arity n whose strategy list does not contain i, the term $f(u_1, \ldots, u_n)$ may be in normal form even though u_i is not. This leads to a characterization of the reducible positions in a term, called *LS-positions* and used in the narrowing step and in the computation of the usable rules.

A complete description of the proof process is given in [9], and its implementation is done in the same way as in the innermost case, respecting the particular technical aspects described above. Although our proof mechanism for local strategies can be seen as a generalization of the one for the innermost case, a specific implementation of the inference rules remains more efficient in the latter case : the computation of the usable rules is simpler, collecting the narrowing mgus to compute disequations can be avoided and the detection of normal forms is simpler, since there is no notion of LS-positions.

6. EXAMPLES AND RELATED WORK

We have studied the behaviour of Cariboo on a collection of examples for the innermost case and local strategies as well, available at

http://www.loria.fr/~fissore/Demo/description.html.

These examples are not terminating for standard rewriting. As regard to the behaviour of the proof process for the innermost case, among the 26 successful examples :

- 11 examples (42 %) are fully automatic and proved to be innermost terminating without any constraint ordering ;
- 12 examples (46 %) are proved to be innermost terminating thanks to simple ordering constraints automatically solved by tools such as C*i*ME2 [8] ;

Table 1: A (compared) collection of innermost terminating TRS

No	TRS	DP method	Cariboo
1	$f(0,1,x) \to f(x,x,x)$ $g(x,y) \to x$ $g(x,y) \to y$	-	-
2	$f(g(x),s(0),y) \to f(y,y,g(x))$ $g(s(x)) \to s(g(x))$ $g(0) \to 0$	$G(s(x)) \succ G(x)$	Usable rules : $g(s(x)) \to s(g(x))$ $g(0) \to 0$
3	$f(g(x,y),x,z) \to f(z,z,z)$ $g(x,y) \to x$ $g(x,y) \to y$	-	-
4	$f(g(x),x,y) \to f(y,y,g(y))$ $g(g(x)) \to g(x)$	$G(g(x)) \succ G(x)$	-
5	$f(0) \to f(0)$ $0 \to 1$	-	-
6	$f(0,1,x) \to f(g(x,x),x,x)$ $g(x,y) \to x$ $g(x,y) \to y$	narrowing	**sat. of** A
7	$f(s(x)) \to g(g(x,x))$ $g(0,1) \to s(0)$ $0 \to 1$	-	-
8	$x+0 \to x$ $x+s(y) \to s(x+y)$ $f(0,s(0),x) \to f(x,x+x,x)$ $g(x,y) \to x$ $g(x,y) \to y$	$PLUS(x,s(y)) \succ PLUS(x,y)$ + narrowing	Usable rules : $plus(x,0) \succ x$ $plus(x,s(y)) \succ s(plus(x,y))$ + **sat. of** A
9	$x+0 \to x$ $x+s(y) \to s(x+y)$ $double(x) \to x+x$ $f(0,s(0),x) \to f(x,double(x),x)$ $g(x,y) \to x$ $g(x,y) \to y$	narrowing	Usable rules : $double(x) \to plus(x,x)$ $plus(x,0) \to x$ $plus(x,s(y)) \to s(plus(x,y))$ + **sat. of** A
10	$f(x,g(x)) \to f(1,g(x))$ $g(1) \to g(0)$	-	-
11	$h(x,z) \to f(x,s(x),z)$ $f(x,y,g(x,y)) \to h(0,g(x,y))$ $g(0,y) \to 0$ $g(x,s(y)) \to g(x,y)$	$G(x,s(y)) \succ G(x,y)$	Usable rules : $g(x,s(y)) \to g(x,y)$ $g(0,y) \to 0$
12	$h(0,x) \to f(0,x,x)$ $f(0,1,x) \to h(x,x)$ $g(x,y) \to x$ $g(x,y) \to y$	$H(0,x) \succeq F(0,x,x)$ $F(0,1,x) \succ H(x,x)$	-
13	$f(0,1,x) \to f(x,x,x)$ $f(x,y,z) \to 2$ $0 \to 2$ $1 \to 2$ $g(x,x,y) \to y$ $g(x,y,y) \to x$	-	-

Table 2: A (compared) collection of innermost terminating TRS

No	TRS	DP method	Cariboo
14	$f(g(x), s(0)) \to f(g(x), g(x))$ $g(s(x)) \to s(g(x))$ $g(0) \to 0$	$G(s(x)) \succ G(x)$ -	Usable rules : $g(s(x)) \succ s(g(x))$ $g(0) \succ 0$
15	$f(0, 1, g(x, y), z) \to$ $f(g(x, y), g(x, y), g(x, y), h(x))$ $g(0, 1) \to 0$ $g(0, 1) \to 1$ $h(g(x, y)) \to h(x)$	expertise (narrowing fails)	Usable rules : $h(g(x, y)) \succ h(x)$ + **sat. of** A
16	$f(s(0), g(x)) \to f(x, g(x))$ $g(s(x)) \to g(x)$	$G(s(x)) \succ G(x)$	Usable rules : $g(s(x)) \succ g(x)$
17	$f(g(x), s(0), y) \to f(g(s(0)), y, g(x))$ $g(s(x)) \to s(g(x))$ $g(0) \to 0$	$G(s(x)) \succ G(x)$ + narrowing	Usable rules : $g(s(x)) \succ s(g(x))$ $g(0) \succ 0$ + **sat. of** A
18	$a(b(a(b(x)))) \to b(a(b(a(a(b(x))))))$	$A(b(a(b(x)))) \succ A(b(x))$	-
19	$f(f(x)) \to f(x)$ $g(0) \to g(f(0))$	modularity + elimination	-
20	$f(1) \to f(g(1))$ $f(f(x)) \to f(x)$ $g(0) \to g(f(0))$ $g(g(x)) \to g(x)$	elimination	-
21	$quot(0, s(y), s(z)) \to 0$ $quot(s(x), s(y), z) \to quot(x, y, z)$ $quot(x, 0, s(z)) \to s(quot(x, s(z), s(z)))$	$QUOT(s(x), s(y), z) \succ QUOT(x, y, z)$ $QUOT(x, 0, s(z)) \succeq QUOT(x, s(z), s(z))$	$\perp$
22	$0 + y \to y$ $s(x) + y \to s(x + y)$ $quot(x, 0, s(z)) \to s(quot(x, z + s(0), s(z)))$	expertise	Usable rules : $0 + y \to y$ $s(x) + y \to s(x + y)$ $+ TERMIN$
23	$intlist(nil) \to nil$ $intlist(x.y) \to s(x).intlist(y)$ $int(0, 0) \to 0.nil$ $int(0, s(y)) \to 0.int(s(0), s(y))$ $int(s(x), 0) \to nil$ $int(s(x), s(y)) \to intlist(int(x, y))$	$INTLIST(x.y) \succ INTLIST(y)$ $INT(0, s(y)) \succeq INT(s(0), s(y))$ $INT(s(x), s(y)) \succ INT(x, y)$	$\perp$
24	$f(x, x) \to f(g(x), x)$ $g(x) \to s(x)$	narrowing	Usable rules : $g(x) \to s(x)$ + **sat. of** A
25	$p(0) \to 0$ $p(s(x)) \to x$ $le(0, y) \to true$ $le(s(x), 0) \to false$ $le(s(x), s(y)) \to le(x, y)$ $minus(x, 0) \to x$ $minus(x, s(y)) \to$ $if(le(x, s(y)), 0, p(minus(x, p(s(y)))))$ $if(true, x, y) \to x$ $if(false, x, y) \to y$	narrowing + RPO	$\perp$

Table 3: A (compared) collection of innermost terminating TRS

No	TRS	DP method	Cariboo
26	$f(x, c(y)) \to f(x, s(f(y, y)))$ $f(s(x), s(y)) \to f(x, s(c(s(y))))$	modularity	$\perp$
27	$f(f(x)) \to f(f(x))$ $f(a) \to a$	$\perp$	**sat. of** A
28	$f(x, y) \to g(g(x, y), y)$ $g(x, y) \to h(1)$ $g(g(x, y), y) \to f(x, y)$	-	$TERMIN$
29	$f(s(x)) \to f(g(h(x)))$ $g(h(x)) \to g(x)$ $g(0) \to s(0)$ $h(0) \to 1$		Usable rules : $g(h(x)) \to g(x)$ $g(0) \to s(0)$ $h(0) \to 1$ + **sat. of** A
30	$half(0) \to 0$ $half(s(0)) \to 0$ $half(s(s(x))) \to s(half(x))$ $log(0) \to 0$ $log(s(0)) \to 0$ $log(s(s(x))) \to s(log(s(half(x))))$	$LOG(s(s(x)))$ $\succ$ $LOG(s(half(x)))$ $half(0) \succeq 0$ $half(s(0)) \succeq 0$ $half(s(s(x)))$ $\succeq s(half(x))$	$\perp$
31	$zero(sharp) \to sharp$ $plus(x, sharp) \to x$ $plus(sharp, x) \to x$ $plus(zero(x), zero(y)) \to zero(plus(x, y))$ $plus(zero(x), one(y)) \to one(plus(x, y))$ $plus(one(x), zero(y)) \to one(plus(x, y))$ $plus(one(x), one(y)) \to zero(plus(plus(x, y), one(sharp)))$ $plus(x, plus(y, z)) \to plus(plus(x, y), z)$ $minus(x, sharp) \to x$ $minus(sharp, x) \to sharp$ $minus(zero(x), zero(y)) \to zero(minus(x, y))$ $minus(zero(x), one(y)) \to one(minus(minus(x, y), one(sharp)))$ $minus(one(x), zero(y)) \to one(minus(x, y))$ $minus(one(x), one(y)) \to zero(minus(x, y))$ $not(false) \to true$ $not(true) \to false$ $if(true, x, y) \to x$ $if(false, x, y) \to y$ $ge(zero(x), zero(y)) \to ge(x, y)$ $ge(zero(x), one(y)) \to not(ge(y, x))$ $ge(one(x), zero(y)) \to ge(x, y)$ $ge(one(x), one(y)) \to ge(x, y)$ $ge(x, sharp) \to true$ $ge(sharp, one(x)) \to false$ $ge(sharp, zero(x)) \to ge(sharp, x)$		Usable rules : $minus$ $+$ $zero$ $plus$ $+$ $zero$ ge

- 8 examples (30 %) require a satisfiability test of the abstraction constraints set A.

On these examples, when possible, we have compared our method with the dependency pair (DP) approach [2]. This comparison is given in Tables 1, 2 and 3, where '_' denotes no constraint solving at all and '$\perp$' means failure.

Some examples raise the same kinds of ordering constraints, but there are also examples where the constraints are different. Indeed, there are cases in which the DP method requires fewer constraint solving or expertise than ours, and some others in which our method concludes in a simpler way. For instance, on the TRS 12, simple constraints automatically verified with the subterm property of the induction ordering are generated by our proof process, while a polynomial interpretation is needed to solve those generated by the dependency pair approach.

Likewise, the TRS 19 is automatic with Cariboo and does not require any constraint solving, while it requires a modular approach and an argument elimination with the dependency pair method.

As for the TRS 27, since it is not innermost terminating on the free term algebra, the DP method does not apply. Cariboo succeeds with a delegated feature on the satisfiability of A.

Our method does not manage the TRSs 23, 25 and 30 yet. However, we are working on a generalization of the inductive process that will enable to handle them. The principle is to recursively apply the inference rules on terms that cannot be handled with the induction hypothesis or with the predicate $TERMIN$.

For a comparison of our proof method for LS-strategies with the context sensitive approach, see [9, 17].

7. CONCLUSION

In this paper, we have presented Cariboo, a tool for proving termination of term rewriting under strategies. Since the inductive process it implements simulates the rewriting relation, it can be applied to different strategies in quite a natural way. The implementation for both innermost and local strategies is now available, and an implementation for the outermost strategy is under development.

The important point to automate our proof principle is the satisfaction of the abstraction and ordering constraints. One of the advantages of our proof process is that its correctness is ensured even though we get rid of these constraints ; by suppressing as many constraints as possible, the proof process is automatic on many examples and less costly than approaches always requiring constraint solving.
When they are considered, ordering constraints can be delegated to existing ordering constraint solvers. Besides, on many examples, the satisfaction of the ordering constraints is immediate : as they only express the subterm property, they are trivially satisfied by any simplification ordering, so we do not need any ordering constraint solver.

We are now integrating Cariboo in a toolbox for validating rule based programs. It is already used in connection with a procedure verifying sufficient completeness, and other termination tools. Its cooperation with completion under strategies and complexity computations is under study.

8. REFERENCES

[1] T. Arts and J. Giesl. Proving innermost normalisation automatically. In *Proceedings 8th Conference on Rewriting Techniques and Applications, Sitges (Spain)*, volume 1232 of *Lecture Notes in Computer Science*, pages 157–171. Springer-Verlag, 1997.

[2] T. Arts and J. Giesl. A collection of examples for termination of term rewriting using dependency pairs. Technical Report AIB-2001-09, RWTH Aachen, Germany, September 2001.

[3] P. Borovanský, C. Kirchner, H. Kirchner, P.-E. Moreau, and Ch. Ringeissen. An Overview of ELAN. In C. Kirchner and H. Kirchner, editors, *Proc. Second Intl. Workshop on Rewriting Logic and its Applications*, Electronic Notes in Theoretical Computer Science, Pont-à-Mousson (France), September 1998. Elsevier.

[4] Peter Borovanský, Claude Kirchner, Hélène Kirchner, Pierre-Etienne Moreau, and Christophe Ringeissen. An overview of ELAN. In Claude Kirchner and Hélène Kirchner, editors, *Proceedings of the second International Workshop on Rewriting Logic and Applications*, volume 15, `http://www.elsevier.nl/locate/entcs/volume15.html`, Pont-à-Mousson (France), September 1998. Electronic Notes in Theoretical Computer Science. Report LORIA 98-R-316.

[5] M. Clavel, S. Eker, P. Lincoln, and J. Meseguer. Principles of Maude. In J. Meseguer, editor, *Proceedings of the 1st International Workshop on Rewriting Logic and its Applications*, volume 5 of *Electronic Notes in Theoretical Computer Science*, Asilomar, Pacific Grove, CA, USA, September 1996. North Holland.

[6] Evelyne Contejean, Claude Marché, Ana-Paula Tomás, and Xavier Urbain. Solving termination constraints via finite domain polynomial interpretations. Unpublished draft, 2000.

[7] S. Eker. Term rewriting with operator evaluation strategies. In C Kirchner and H. Kirchner, editors, *Proceedings of the 2nd International Workshop on Rewriting Logic and its Applications*, Pont-à-Mousson, France, September 1998.

[8] Benjamin Monate et Xavier Urbain
Evelyne Contejean, Claude Marché. Cime version 2, 2000. Version préliminaire disponible à http://cime.lri.fr/.

[9] O. Fissore, I. Gnaedig, and H. Kirchner. Termination of rewriting with local strategies. In M. P. Bonacina and B. Gramlich, editors, *Selected papers of the 4th International Workshop on Strategies in Automated Deduction*, volume 58 of *Electronic Notes in Theoretical Computer Science*. Elsevier Science Publishers, 2001. Also available as Technical Report A01-R-177, LORIA, Nancy, France.

[10] K. Futatsugi and A. Nakagawa. An overview of CAFE specification environment – an algebraic approach for creating, verifying, and maintaining formal specifications over networks. In *Proceedings of the 1st IEEE Int. Conference on Formal Engineering Methods*, 1997.

[11] J. Giesl and Middeldorp A. Transforming

Context-Sensitive Rewrite Systems. In *Proceedings of the 10th International Conference on Rewriting Techniques and Applications*, volume 1631 of *Lecture Notes in Computer Science*, pages 271–285, Trento, Italy, 1999. Springer-Verlag.

[12] Jurgen Giesl. Polo – a system for termination proofs using polynomial orderings. Technical Report IBN 95/24, Technische Hochschule Darmstadt, 1995.

[13] I. Gnaedig, H. Kirchner, and O. Fissore. Induction for innermost and outermost ground termination. Technical Report A01-R-178, LORIA, Nancy, France, 2001.

[14] J. Goguen, T. Winkler, J. Meseguer, K. Futatsugi, and J.P. Jouannaud. Introducing OBJ3. Technical report, Computer Science Laboratory, SRI International, march 1992.

[15] P. Klint. A meta-environment for generating programming environments. *ACM Transactions on Software Engineering and Methodology*, 2:176–201, 1993.

[16] M.R.K. Krishna Rao. Some characteristics of strong normalization. *Theoretical Computer Science*, 239:141–164, 2000.

[17] S. Lucas. Termination of rewriting with strategy annotations. In A. Voronkov and R. Nieuwenhuis, editors, *Proc. of 8th International Conference on Logic for Programming, Artificial Intelligence and Reasoning, LPAR'01*, volume 2250 of *Lecture Notes in Artificial Intelligence*, pages 669–684, La Habana, Cuba, December 2001. Springer-Verlag, Berlin.

[18] H. Zantema. Termination of context-sensitive rewriting. In *Proceedings of the 8th International Conference on Rewriting Techniques and Applications*, volume 1232 of *Lecture Notes in Computer Science*, pages 172–186. Springer-Verlag, 1997.

APPENDIX

A. SOME PARTS OF THE ELAN CODE

Recall the reflexive aspect of our termination tool : for proving termination of rules based programs, we have proposed an inference rule based mechanism, directly translated in the rule based programming language ELAN. Here are some important Elan rules extracted from our code : the inference rules **Abstract** and **Narrow** together with the computation of the usable rules.

A.1 The rule Narrow

The inference rule **Narrow** is coded in ELAN by the following labelled rule :

```
  [Narrow] [t_ref,[t,A,C],[depth,no_var]]
           => [t_ref,[t_res,AA,C],[depth+1,no_var+nb]]
where OK := (cont) neq_term(t,vide)
                              and neq_term(t,bottom)
where rwr := (listExtract) elem(rwrules)
where r := () second(rwr)
where l := () first(rwr)
where omega := (chooseOccurence) nvocc(t)
where match := (unifys) true & t at omega = l
where theta1 := () system_to_subst(match)
where theta2 := () rename_Subst(varlist_Vars, match,
                                                no_var)

where matchaf := () eqs_to_af(apply(theta2, match))

where nb := () nb_var_in(varlist_Vars,match)
where theta := () theta1 o theta2
where t_res := () theta(t[r] at omega)
where AA := (simplify) apply(theta,A)
where ch2 := () "(For Narrow) "
where OK1 := (cont) cond_narrow(ch2, AA,C,false)
where ch1 := () "[ Narrow ]"
where pid_1 := () ecrire(ch1, t_res, matchaf, AA, C,
                                                 depth)
where pid_2 := () close(pid_1)
  end
```

We see that the rule **Narrow** transforms the triple $[t, A, C]$ into $[t\_res, AA, C]$, where :

- *t_res* is the result of the narrowing step of t at the top position (denoted *omega* in ELAN) and with the narrowing mgu *theta*, given by the ELAN affectation :

  ```
  where t_res := () theta(t[r] at omega).
  ```

- AA is the transformation of A :

  ```
  where AA := (simplify) apply(theta,A).
  ```

The variables *depth* stands for the current depth of the derivation tree. It is used to limit the size of the derivation tree, by comparison with a maximum value. The variable *no_var* stands for the current number of fresh NF-variables, used to generate new ones.

A.2 The rule Abstract

The inference rule **Abstract** is coded in ELAN by the following labelled rule :

```
[Abstract] [t_ref,[F(v_1,...,v_N),A,C],[depth,no_Var]]
                  => [t_ref,[t_res,AA,CC],[depth+1,nb]]
```

```
where res := (dk(weak)) cond_abstract(A,C,t_ref,
                              F(v_1,...,v_N),no_Var, depth)

where OK := (cont) 1-th(res)
where AA := (simplify) 2-th(res)
where CC := () 3-th(res)
where t_res := () 4-th(res)
where nb := () 5-th(res)

// Ecriture du resultat
where chaine := () "[ Abstract ]"
where pid1 := () ecrire(chaine, t_res, AA, CC, depth)
where pid2 := () close(pid1)
  end
```

Like the rule **Narrow**, it transforms a triple $[t, A, C]$ where $t = F(v\_1, \ldots, v\_N)$. Note the convenient ELAN notation "...", enabling to avoid the exhaustive enumeration of the arguments of F. The purpose of the predicate *cond_abstract* is to abstract, if need be, all subterms $v\_1, \ldots, v\_N$.

A.3 The usable rules

Let us first recall the definition of the usable rules. Let $\mathcal{R}$ be a TRS on a set $\mathcal{F}$ of symbols. Let $Rls(f) = \{l \to r \in \mathcal{R} \mid root(l) = f\}$, where $root(l)$ is the top symbol of the term l. For any $t \in \mathcal{T}(\mathcal{F}, \mathcal{X} \cup \mathcal{N})$, the set of usable rules of t, denoted $\mathcal{U}(t)$, is defined by:

- $\mathcal{U}(t) = \mathcal{R}$ if $t \in \mathcal{X} \setminus Var(\mathcal{R})$,
- $\mathcal{U}(t) = \emptyset$ if $t \in \mathcal{N} \cup Var(\mathcal{R})$,
- $\mathcal{U}(f(u_1, \ldots, u_n)) = Rls(f) \cup \bigcup_{i=1}^{n} \mathcal{U}(u_i) \cup \bigcup_{l \to r \in Rls(f)} \mathcal{U}(r)$,

where $Var(\mathcal{R})$ denotes the set of variables of $\mathcal{R}$. Here is (a part of) the ELAN program encoding the computation of the usable rules, where, given a TRS R, $U(t, R)$ computes the usable rules of a term t by using *Ucalc* :

```
  [] U(t,R) => R1 if not liste_vide(R)
       where R1 := (first(var, Var, terme)) Ucalc(t,R)
  end

  [] U(t,R) => nil_pair[term,term] if liste_vide(R) end

  [var] Ucalc(var,R) => R if not estVar(var) end

  [Var] Ucalc(var,R) => nil_pair[term,term]
                          if estVar(var) end

  [terme] Ucalc(F(u_1,...,u_N),R) => res
                         if not liste_vide(R)
where liste := () to_LS_INN(F)
where R1 := () Rls(head(F(u_1,...,u_N)),R)
where R3 := () setminus(R,R1)
where UdesRhs := () Ucalc_rhs(R1, R3)
where R2 := () setminus(R3,UdesRhs)

where UdesUi := () Ucalc_rec(F(u_1,...,u_N), liste, R2)
where res := () appendset(appendset(R1,UdesRhs),UdesUi)
  end

  [terme] Ucalc(F(u_1,...,u_N),R) => nil_pair[term,term]
                         if liste_vide(R) end
```

Note the three-cases behavior of the *Ucalc* operator, that computes the usable rules of a term accordingly to its form :

1. if t is a variable of $\mathcal{X}$, then the whole system is returned ;
2. if t is a NF-variable (predicate $estVar(t)$ in the condition of the rule), then it has no usable rules ;
3. otherwise, the set of rewrite rules $Rls(F)$ is computed along with the usable rules of their right-hand sides. Then *Ucalc* is recursively applied on each *u_i*.